DTN Routing as a Resource Allocation Problem

Aruna Balasubramanian Brian Neil Levine Arun Venkataramani
Department of Computer Science, University of Massachusetts, Amherst, USA 01003
{arunab, brian, arun}@cs.umass.edu

revised: October 25, 2009

ABSTRACT

Routing protocols for disruption-tolerant networks (DTNs) use a variety of mechanisms, including discovering the meeting probabilities among nodes, packet replication, and network coding. The primary focus of these mechanisms is to increase the likelihood of finding a path with limited information, and so these approaches have only an *incidental* effect on such routing metrics as maximum or average delivery delay. In this paper, we present RAPID, an *intentional* DTN routing protocol that can optimize a specific routing metric such as the worst-case delivery delay or the fraction of packets that are delivered within a deadline. The key insight is to treat DTN routing as a resource allocation problem that translates the routing metric into per-packet utilities which determine how packets should be replicated in the system. We evaluate RAPID rigorously through a prototype deployed over a vehicular DTN testbed of 40 buses and simulations based on real traces. To our knowledge, this is the first paper to report on a routing protocol deployed on a real outdoor DTN. Our results suggest that RAPID significantly outperforms existing routing protocols for several metrics. We also show empirically that for small loads, RAPID is within 10% of the optimal performance.

1. INTRODUCTION

Disruption-tolerant networks (DTNs) enable transfer of data when mobile nodes are connected only intermittently. Applications of DTNs include large-scale disaster recovery networks, sensor networks for ecological monitoring [30], ocean sensor networks [22, 19], vehicular networks [21, 6], and projects such as TIER [2], Digital Study Hall [12], and One Laptop Per Child [1] to benefit developing nations. Intermittent connectivity can be a result of mobility, power management, wireless range, sparsity, or malicious attacks. The inherent uncertainty about network conditions make routing in DTNs a challenging problem.

The primary focus of many existing DTN routing protocols is to increase the likelihood of finding a path with extremely limited information. To discover such a path, a variety of mechanisms are used, including estimating node meeting probabilities, packet replication, network coding, placement of stationary waypoint stores, and leveraging prior knowledge of mobility patterns. Unfortunately, the burden of finding even one path is so great that existing approaches have only an *incidental* rather than an *intentional* effect on such routing *metrics* as worst-case delivery latency, average delay, or percentage of packets delivered. This disconnect between application needs and routing protocols hinders deployment of DTN applications. Currently, it is difficult to drive the routing layer of a DTN by specifying priorities, deadlines, or cost constraints. For example, a simple news and information application is better served by maximizing the number of news stories delivered before they are outdated, rather than eventually delivering all stories.

In this paper, we formulate the DTN routing problem as a *resource allocation problem*. The protocol we describe, called RAPID (Resource Allocation Protocol for Intentional DTN) routing, allocates resources to packets to optimize an administrator-specified routing metric. At each transfer opportunity, a RAPID node replicates or allocates bandwidth resource to a set of packets in its buffer, in order to optimize the given routing metric. Packets are delivered through opportunistic replication, until a copy reaches the destination.

RAPID makes the allocation decision by first translating the routing metric to a per-packet utility. DTNs are resource-constrained networks in terms of transfer bandwidth, energy, and storage; allocating resources to replicas without careful attention to available resources can cause more harm than good. Therefore, a RAPID node replicates packets in the order of their marginal utility of replication, i.e., the first packet to be replicated is the one that provides the highest increase in utility per unit resource used. We show how RAPID can use this simple approach to optimize three different routing metrics: average delay, worst-case delay, and the number of packets delivered before a deadline.

RAPID loosely tracks network resources through a control plane to assimilate a local view of the global network state. To this end, RAPID uses an in-band *control channel* to exchange network state information among nodes using a fraction of the available bandwidth, and uses the additional information to significantly improve routing performance. RAPID's control channel builds on insights from previous work. For example, Jain et al. [15] suggest that DTN routing protocols that use more knowledge of

network conditions perform better, and Burgess et al. [6] show that flooding acknowledgments improves delivery rates by removing useless packets from the network.

We present hardness results to substantiate RAPID's heuristic approach. We prove that online algorithms without complete future knowledge and with unlimited computational power, or computationally limited algorithms with complete future knowledge, can be arbitrarily far from optimal.

We have built and deployed RAPID on a vehicular DTN testbed, DieselNet [6], that consists of 40 buses covering a 150 square-mile area around Amherst, MA. We collected 58 days of performance traces of the RAPID deployment. To our knowledge, this is the first paper to report on a routing protocol deployed on a real outdoor DTN. Similar testbeds have deployed only flooding as a method of packet propagation [30]. We also conduct a simulation-based evaluation using real traces to stress-test and compare various protocols. We show that the performance results from our trace-driven simulation is within 1% of the real measurements with 95% confidence. We use this simulator to compare RAPID to four existing routing protocols [18, 25, 6] and random routing. We also compare the protocols using synthetic mobility models.

We evaluate the performance of RAPID for three different routing metrics: average delay, worst-case delay, and the number of packets delivered before a deadline. All experiments include the cost of RAPID's control channel. Our experiments using trace-driven and synthetic mobility scenarios show that RAPID significantly outperforms the four routing protocols. For example, in trace-driven experiments under moderate-to-high loads, RAPID outperforms the second-best protocol by about 20% for all three metrics, while also delivering 15% more packets for the first two metrics. With *a priori* mobility information and moderate-to-high loads, RAPID outperforms random replication by about 50% for high packet loads. We also compare RAPID to an optimal protocol and show empirically that RAPID performs within 10% of optimal for low loads.

2. RELATED WORK

Replication versus Forwarding.

We classify related existing DTN routing protocols as those that replicate packets and those that forward only a single copy. *Epidemic* routing protocols replicate packets at transfer opportunities hoping to find a path to a destination. However, naive flooding wastes resources and can severely degrade performance. Proposed protocols attempt to limit replication or otherwise clear useless packets in various ways: *(i)* using historic meeting information [11, 7, 6, 18]; *(ii)* removing useless packets using acknowledgments of delivered data [6]; *(iii)* using probabilistic mobility information to infer delivery [24]; *(iv)* replicating packets with a small probability [29]; *(v)* using network coding [28] and coding with redundancy [14]; and *(vi)* bounding the number of replicas of a packet [25, 24, 20].

In contrast, *forwarding* routing protocols maintain at most one copy of a packet in the network [15, 16, 27]. Jain et al. [15] propose a forwarding algorithm to minimize the average delay of packet delivery using oracles with varying degrees of future knowledge. Our deployment experience suggests that, even for a scheduled bus service, implementing the simplest oracle is difficult; connection opportunities are affected by many factors in practice including weather, radio interference, and system failure. Furthermore, we present formal hardness and empirical results to quantify the impact of not having complete knowledge.

Jones et al. [16] propose a link-state protocol based on epidemic propagation to disseminate global knowledge, but use a single path to forward a packet. Shah et al. [23] and Spyropoulos et al. [27] present an analytical framework for the forwarding-only case assuming a grid-based mobility model. They subsequently extend the model and propose a replication-based protocol, Spray and Wait [25]. The consensus appears to be [25] that replicating packets can improve performance (and security [5]) over just forwarding, but risk degrading performance when resources are limited.

Incidental versus Intentional.

Our position is that most existing schemes only have an *incidental* effect on desired performance metrics, including commonly evaluated metrics such as average delay or delivery probability. Therefore, the effect of a routing decision on the performance of a given resource constrained network scenario is unclear. For example, several existing DTN routing algorithms [25, 24, 20, 6] route packets using the number of replicas as the heuristic, but the effect of replication varies with different routing metrics. *Spray and Wait* [25] routes to reduce delay metric, but it does not take into account bandwidth or storage constraints. In contrast, routing in RAPID is *intentional* with respect to a given performance metric. RAPID explicitly calculates the effect of replication on the routing metric while accounting for resource constraints.

Resource Constraints.

RAPID also differs from most previous work in its assumptions regarding resource constraints, routing policy, and mobility patterns. Table 1 shows a taxonomy of many existing DTN routing protocols based on assumptions about *bandwidth* available during transfer opportunities and the *storage* carried by nodes; both are either *finite* or *unlimited*. For each work, we state

Problem	Storage	Bandwidth	Routing	Previous work (and mobility)
P1	Unlimited	Unlimited	Replication	Epidemic [20], Spray and Wait [25]: Constraint in the form of channel contention (Grid-based synthetic)
P2	Unlimited	Unlimited	Forwarding	Modified Djikstra's et al. [15] (simple graph), MobySpace [17] (Powerlaw)
P3	Finite	Unlimited	Replication	Davis et al. [11] (Simple partitioning synthetic), SWIM [24] (Exponential), MV [7] (Community-based synthetic), Prophet [18] (Community-based synthetic)
P4	Finite	Finite	Forwarding	Jones et al. [16] (AP traces), Jain et al. [15] (Synthetic DTN topology)
P5	Finite	Finite	Replication	This paper (Vehicular DTN traces, exponential, and power law meeting probabilities, testbed deployment), MaxProp [6] (Vehicular DTN traces)

Table 1: A classification of some related work into DTN routing scenarios

in parentheses the mobility model used. RAPID is a replication-based algorithm that assumes constraints on both storage and bandwidth (P5) — the most challenging and most practical problem space.

P1 and P2 are important to examine for valuable insights that theoretical tractability yields but are impractical for real DTNs with limited resources. Many studies [18, 11, 7, 24] analyze the case where storage at nodes is limited, but bandwidth is unlimited (P3). However, we find this scenario to be uncommon. Bandwidth is likely to be constrained for most typical DTN scenarios. Specifically, in mobile and vehicular DTNs, transfer opportunities are typically short-lived [13, 6].

We were unable to find other protocols in P5 except *MaxProp* [6] that assume limited storage and bandwidth. However, it is unclear how to optimize a specific routing metric using *MaxProp*, so we categorize it as an incidental routing protocol. Our experiments indicate that RAPID outperforms *MaxProp* for each metric that we evaluate.

Some theoretical works [31, 26, 24, **?**] derive closed-form expressions for average delay and number of replicas in the system as a function of the number of nodes and mobility patterns. Although these analyses contributed to important insights in the design of RAPID, their assumptions about mobility patterns or unlimited resources were, in our experience, too restrictive to be applicable to practical settings.

3. THE RAPID PROTOCOL

3.1 System model

We model a DTN as a set of mobile nodes. Two nodes transfer data packets to each other when within communication range. During a transfer, the sender replicates packets while retaining a copy. A node can deliver packets to a destination node directly or via intermediate nodes, but packets may not be fragmented. There is limited storage and transfer bandwidth available to nodes. Destination nodes are assumed to have sufficient capacity to store delivered packets, so only storage for in-transit data is limited. Node meetings are assumed to be short-lived. The nodes are assumed to have sufficient computational capabilities as well as enough resources to maintain state information.

Formally, a DTN consists of a node meeting schedule and a workload. The node meeting schedule is a directed multigraph $G = (V, E)$, where V and E represent the set of nodes and edges, respectively. Each directed edge e between two nodes represents a meeting between them, and it is annotated with a tuple (t_e, s_e), where t is the time and s is the size of the transfer opportunity. The workload is a set of packets $P = \{(u_1, v_1, s_1, t_1), (u_2, v_2, s_2, t_2), \ldots\}$, where the ith tuple represents the source, destination, size, and time of creation (at the source), respectively, of packet i. The goal of a DTN routing algorithm is to deliver all packets using a feasible schedule of packet transfers, where *feasible* means that the total size of packets transfered during each opportunity is less than the size of the opportunity, always respecting storage constraints.

In comparison to Jain et al.[15] who model link properties as continuous functions of time, our model assumes discrete short-lived transfers; this makes the problem analytically more tractable and characterizes many practical DTNs well.

3.2 RAPID design

RAPID models DTN routing as a utility-driven resource allocation problem. A packet is routed by replicating it until a copy reaches the destination. The key question is: given limited bandwidth, how should packets be replicated in the network so as to optimize a specified *routing metric*? RAPID derives a per-packet *utility function* from the routing metric. At a transfer opportunity, it replicates a packet that locally results in the highest increase in utility.

Consider a routing metric such as *minimize average delay of packets*, the running example used in this section. The corresponding utility U_i of packet i is the negative of the expected delay to deliver i, i.e., the time i has already spent in the system plus the additional expected delay before i is delivered. Let δU_i denote the increase in U_i by replicating i and s_i denote the size of i. Then, RAPID replicates the packet with the highest value of $\delta U_i / s_i$ among packets in its buffer; in other words, the

$D(i)$	Packet i's expected delay = $T(i) + A(i)$
$T(i)$	Time since creation of i
$a(i)$	Random variable that determines the remaining time to deliver i
$A(i)$	Expected remaining time = $\mathrm{E}[a(i)]$
M_{XZ}	Random variable that determines inter-meeting time between nodes X and Z

Table 2: List of commonly used variables.

packet with the highest marginal utility.

In general, U_i is defined as the expected contribution of i to the given routing metric. For example, the metric *minimize average delay* is measured by summing the delay of packets. Accordingly, the utility of a packet is its expected delay. Thus, RAPID is a heuristic based on locally optimizing marginal utility, i.e., the expected increase in utility per unit resource used.

Using the marginal utility heuristic has some desirable properties. The marginal utility of replicating a packet to a node is low when *(i)* the packet has many replicas, or *(ii)* the node is a poor choice with respect to the routing metric, or *(iii)* the resources used do not justify the benefit. For example, if nodes meet each other uniformly, then a packet i with 6 replicas has lower marginal utility of replication compared to a packet j with just 2 replicas. On the other hand, if the peer is unlikely to meet j's destination for a long time, then i may take priority over j.

RAPID has three core components: a *selection* algorithm, an *inference* algorithm, and a *control channel*. The selection algorithm is used to determine *which* packets to replicate at a transfer opportunity given their utilities. The inference algorithm is used to estimate the *utility* of a packet given the routing metric. The control channel propagates the necessary metadata required by the inference algorithm.

3.3 The selection algorithm

The RAPID protocol executes when two nodes are within radio range and have discovered one another. The protocol is symmetric; without loss of generality, we describe how node X determines which packets to transfer to node Y (refer to the box marked PROTOCOL RAPID).

RAPID also adapts to storage restrictions for in-transit data. If a node exhausts all available storage, packets with the lowest utility are deleted first as they contribute least to overall performance. However, a source never deletes its own packet unless it receives an acknowledgment for the packet.

3.4 Inference algorithm

Next, we describe how PROTOCOL RAPID can support specific metrics using an algorithm to infer utilities. Table 2 defines the relevant variables.

PROTOCOL RAPID(X, Y):

1. *Initialization*: Obtain metadata from Y about packets in its buffer as well as metadata it collected over past meetings (detailed in Section 4.2).
2. *Direct delivery*: Deliver packets destined to Y in decreasing order of creation times.
3. *Replication*: For each packet i in node X's buffer
   (a) If i is already in Y's buffer (as determined from the metadata), ignore i.
   (b) Estimate marginal utility, $\delta U_i/s_i$, of replicating i to Y.
   (c) Replicate packets in decreasing order of marginal utility.
4. *Termination*: End transfer when out of radio range or all packets replicated.

3.4.1 Metric 1: Minimizing average delay

To minimize the average delay of packets in the network we define the utility of a packet as

$$U_i = -D(i) \tag{1}$$

since the packet's expected delay is its contribution to the performance metric. RAPID attempts to greedily replicate the packet whose replication reduces the delay by the most among all packets in its buffer.

3.4.2 Metric 2: Minimizing missed deadlines

To minimize the number of packets that miss their deadlines, the utility is defined as the probability that the packet will be delivered within its deadline:

$$U_i = \begin{cases} P(a(i) < L(i) - T(i)), & L(i) > T(i) \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where $L(i)$ is the packet life-time. A packet that has missed its deadline can no longer improve performance and is thus assigned a value of 0. The marginal utility is the improvement in the probability that the packet will be delivered within its deadline.

3.4.3 Metric 3: Minimizing maximum delay

To minimize the maximum delay of packets in the network, we define the utility U_i as

$$U_i = \begin{cases} -D(i), & D(i) \geq D(j) \quad \forall j \in S \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where S denotes the set of all packets in X's buffer. Thus, U_i is the negative expected delay if i is a packet with the maximum expected delay among all packets held by Y. So, replication is useful only for the packet whose delay is maximum. For the routing algorithm to be work conserving, RAPID computes utility for the packet whose delay is currently the maximum; i.e., once a packet with maximum delay is evaluated for replication, the utility of the remaining packets is recalculated using Eq. 3.

4. ESTIMATING DELIVERY DELAY

How does a RAPID node estimate expected delay in Eqs. 1 and 3, or the probability of packet delivery within a deadline in Eq. 2? The expected delivery delay is the minimum expected time until any node with the replica of the packet delivers the packet; so a node needs to know which other nodes possess replicas of the packet and when they expect to meet the destination.

To estimate expected delay we assume that each node with the copy of the packet delivers the packet directly to the destination, ignoring the effect of further replications. This assumption simplifies the expected delay estimation, and we make this assumption only for networks with dense node meetings, were every node meets every other node. In Section 4.1.2, we describe a modification to this assumption for networks with sparse node meetings. Estimating expected delay is nontrivial even with an accurate global snapshot of system state. For ease of exposition, we first present RAPID's estimation algorithm as if we had knowledge of the global system state, and then we present a practical distributed implementation.

4.1 Algorithm Estimate_Delay

A RAPID node uses the algorithm ESTIMATE_DELAY to estimate the delay of a packet in its buffer. ESTIMATE_DELAY works as follows (refer to box marked ALGORITHM ESTIMATE_DELAY): In Step 1, each node X maintains a separate queue of packets Q destined to a node Z sorted in decreasing order of creation times; this is the order in which the packets will be delivered when X meets Z in PROTOCOL RAPID. In Step 2 of ESTIMATE_DELAY, X computes the delivery delay distribution of packet i if delivered directly by X. In Step 3, X computes the minimum across all replicas of the corresponding delivery delay distributions; we note that the delivery time of i is the time until the first node delivers the packet. ESTIMATE_DELAY assumes that the meeting time distribution is the same as the inter-meeting time distribution.

The *Assumption 2* in ESTIMATE_DELAY is a simplifying independence assumption that does not hold in general. Consider Figure 2(a), an example showing the positions of packet replicas in the queues of different nodes. All packets have a common destination Z and each queue is sorted by $T(i)$. Assume that the transfer opportunities and packets are of unit-size.

In Figure 2(a), packet b may be delivered in two ways: *(i)* if W meets Z; *(ii)* one of X and Y meets Z and then one of X and Y meet Z again. These delay depen-

ALGORITHM ESTIMATE_DELAY:
Node X storing a set of packets Q to destination Z performs the following steps to estimate the time until packet $i \in Q$ is delivered

1. X sorts all packets $i \in Q$ in the descending order of $T(i)$, time since i is created.

   a) Let $b(i)$ be the sum size of packets that precede packet i in the sorted list of X. Figure 1 illustrates a sorted buffer containing packet i.

   b) Let B be the expected transfer opportunity in bytes between X and Z. (For readability, we drop subscript X since we are only talking about one node; in general $b(i)$ and B are functions of the node). Node X locally computes B as a moving average of past transfers between X and Z.

2. *Assumption 1: Suppose only X delivers packets to Z with no further replication.*

   Let $a_X(i)$ be the delay distribution of X delivering the packet. Under our assumption, X requires $\lceil b(i)/B \rceil$ meetings with Z to deliver i.

   Let M be a distribution that models the inter-meeting times between nodes, and let $M_{X,Z}$ be the random variable that represents the time taken for X and Z to meet. We transform $M_{X,Z}$ to random variable $M'_{X,Z}$ that represents the time until X and Z meet $\lceil b(i)/B \rceil$ times. Then, by definition

   $$a_X(i) = M'_{X,Z} \quad (4)$$

3. *Assumption 2: Suppose the k random variables $a_y(i), y \in [1,k]$ were independent, where k is the number of replicas of i.*

   The probability of delivering i within time t is the minimum of the k random variables $a_y(i), y \in [1,k]$. This probability is:

   $$\mathrm{P}(a(i) < t) = 1 - \prod_{y=1}^{k} (1 - \mathrm{P}(a_y(i) < t) \quad (5)$$

   a) Accordingly: $$A(i) = \mathrm{E}[a(i)] \quad (6)$$

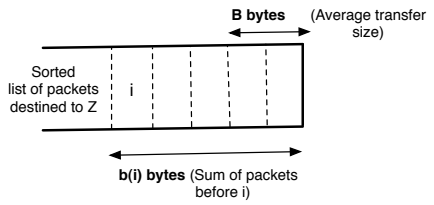

Figure 1: Position of packet i in a queue of packets destined to Z.

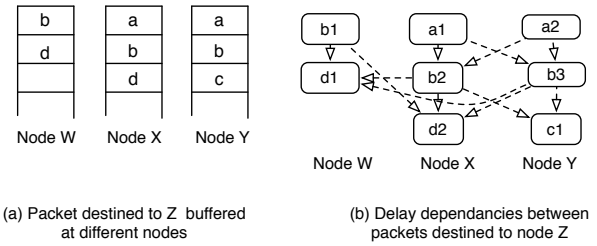

Figure 2: Delay dependencies between packets destined to Z buffered in different nodes.

dencies can be represented using a dependency graph as illustrated in Fig 2(b); packets with the same letter and different indices are replicas. A vertex corresponds to a packet replica. An edge from one node to another indicates a dependency between the delays of the corresponding packets. Recall that M_{XY} is the random variable that represents the meeting time between X and Y.

ESTIMATE_DELAY ignores all the non-vertical dependencies. For example, it estimates b's delivery time distribution as

$$\min(M_{WZ}, M_{XZ} + M_{XZ},\ M_{YZ} + M_{YZ}),$$

whereas the distribution is actually

$$\min(M_{WZ}, \min(M_{XZ}, M_{YZ}) + \min(M_{XZ}, M_{YZ})).$$

Estimating delays without ignoring the non-vertical dependancies is challenging. Using a simplifying assumption that the transfer opportunities and packets are unit-sized, we design algorithm DAG_DELAY(described in a Technical report citerapid-tr), that estimates the expected delay by taking into account non-vertical dependancies. Although DAG_DELAY is of theoretical interest, it cannot be implemented in practice because DAG_DELAY assumes that — *(i)* the transfer opportunity size is exactly equal to the size of a packet.This assumption is fundamental for the design of DAG_DELAYand *(ii)* nodes have a global view of the system.

In general, ignoring non-vertical edges can arbitrarily inflate delay estimates for some pathological cases (detailed in a Technical report [3]). However, we find that ESTIMATE_DELAY works well in practice, and is simple and does not require a global view of the system.

4.1.1 Estimating delays when transfer opportunities are exponentially distributed

We walk through the distributed implementation of ESTIMATE_DELAY for a scenario where the inter-meeting time between nodes is exponentially distributed. Assume that the mean meeting time between nodes is $\frac{1}{\lambda}$. In the absence of bandwidth restrictions, the expected delivery delay when there are k replicas is the mean meeting time divided by k, i.e., $\mathrm{P}(a(i) < t) = 1 - e^{-k\lambda t}$ and $A(i) = \frac{1}{k\lambda}$. (Note that the minimum of k i.i.d. exponentials is also an exponential with mean $\frac{1}{k}$ of the mean of the i.i.d exponentials [8].)

When transfer opportunities are limited, the expected delay depends on the packet's position in the nodes' buffers. In Step 2 of ESTIMATE_DELAY, the node estimates the number of times it needs to meet the destination to deliver a packet as a function of $\lceil b(i)/B \rceil$. According to our exponential meeting time assumption, the time for some node X to meet the destination $\lceil b(i)/B \rceil$ times is described by a gamma distribution with mean $\frac{1}{\lambda} \cdot \lceil b(i)/B \rceil$.

If packet i is replicated at k nodes, Step 3 computes the delay distribution $a(i)$ as the minimum of k gamma variables. We do not know of a closed form expression for the minimum of gamma variables. Instead, if we assume that the time taken for a node to meet the destination $b(i)/B$ times is exponential with the same mean $\frac{1}{\lambda} \cdot \lceil b(i)/B \rceil$. We can then estimate $a(i)$ as the minimum of k exponentials.

Let $n_1(i), n_2(i), \ldots, n_k(i)$ be the number of times each of the k nodes respectively needs to meet the destination to deliver i directly. Then $A(i)$ is computed as:

$$\mathrm{P}(a(i) < t) = 1 - e^{-(\frac{\lambda}{n_1(i)} + \frac{\lambda}{n_2(i)} + \ldots + \frac{\lambda}{n_k(i)})t} \quad (7)$$

$$A(i) = \frac{1}{\frac{\lambda}{n_1(i)} + \frac{\lambda}{n_2(i)} + \ldots + \frac{\lambda}{n_k(i)}} \quad (8)$$

When the meeting time distributions between nodes are non-uniform, say with means $\frac{1}{\lambda_1}, \frac{1}{\lambda_2} \ldots \frac{1}{\lambda_k}$ respectively, then $A(i) = (\frac{\lambda_1}{n_1(i)} + \frac{\lambda_2}{n_2(i)} + \ldots + \frac{\lambda_k}{n_k(i)})^{-1}$.

4.1.2 Estimating delays when transfer opportunity distribution is unknown

To implement RAPID on the DieselNet testbed, we adapt Eq. 8 to scenarios where the transfer opportunities are not exponentially distributed. First, to estimate mean inter-node meeting times in the DieselNet testbed, every node tabulates the average time to meet every other node based on past meeting times. Nodes exchange this table as part of metadata exchanges (Step 1 in PROTOCOL RAPID). A node combines the metadata into a meeting-time adjacency matrix and the information is updated after each transfer opportunity. The matrix

contains the expected time for two nodes to meet directly, calculated as the average of past meetings.

Node X estimates $E(M_{XZ})$, the expected time to meet Z, using the meeting-time matrix. $E(M_{XZ})$ is estimated as the expected time taken for X to meet Z in at most h hops. (Unlike uniform exponential mobility models, some nodes in the trace never meet directly.) For example, if X meets Z via an intermediary Y, the expected meeting time is the expected time for X to meet Y and then Y to meet Z in 2 hops. In our implementation we restrict $h = 3$. When two nodes never meet, even via three intermediate nodes, we set the expected inter-meeting time to infinity. Several DTN routing protocols [6, 18, 7] use similar techniques to estimate meeting probability among peers.

RAPID estimates expected meeting times by taking into account transitive meetings. However, our delivery estimation (described in ESTIMATE_DELAY) assumes that nodes do not make additional replicas. This disconnect is because, in DieselNet, only few buses meet directly, and the pair-wise meeting times between several bus pairs is infinity. We take into account transitive meetings when two buses do not meet directly, to increase the number of potential forwarders.

Let replicas of packet i destined to Z reside at nodes $X_1, \ldots, X_k$. Since we do not know the meeting time distributions, we simply assume they are exponentially distributed. Then from Eq. 8, the expected delay to deliver i is

$$A(i) = [\sum_{j=1}^{k} \frac{1}{E(M_{X_j Z}) \cdot n_j(i)}]^{-1} \quad (9)$$

We use an exponential distribution because bus meeting times in the testbed are difficult to model. Buses change routes several times in one day, the inter-bus meeting distribution is noisy, and we found them hard to model even using mixture models. Approximating meeting times as exponentially distributed makes delay estimates easy to compute and performs well in practice.

4.2 Control channel

Previous studies [15] have shown that as nodes have the benefit of more information about global system state using oracles, they can make significantly better routing decisions. We extend this idea to practical DTNs where no oracle is available. RAPID nodes gather knowledge about the global system state by disseminating metadata using a fraction of the transfer opportunity.

RAPID uses an in-band *control channel* to exchange acknowledgments for delivered packets as well as metadata about every packet learnt from past exchanges. For each encountered packet i, RAPID maintains a list of nodes that carry the replica of i, and for each replica, an estimated time for direct delivery. Metadata for delivered packets is deleted when an ack is received.

For efficiency, a RAPID node maintains the time of last metadata exchange with its peers. The node only sends information about packets whose information changed since the last exchange, which reduces the size of the exchange considerably. A RAPID node sends the following information on encountering a peer: *(i)* Average size of past transfer opportunities; *(ii)* Expected meeting times with nodes; *(iii)* Acks; *(iv)* For each of its own packets, the updated delivery delay estimate based on current buffer state; *(v)* Delivery delay of other packets if modified since last exchange.

When using the control channel, nodes have only an imperfect view of the system. The propagated information may be stale due to changes in number of replicas, changes in delivery delays, or if the packet is delivered but acknowledgments have not propagated. Nevertheless, our experiments confirm that *(i)* this inaccurate information is sufficient for RAPID to achieve significant performance gains over existing protocols and *(ii)* the overhead of metadata itself is not significant.

5. IMPLEMENTATION ON A VEHICULAR DTN TESTBED

We implemented and deployed RAPID on our vehicular DTN testbed, DieselNet [6] (`http://prisms.cs.umass.edu/dome`), consisting of 40 buses, of which a subset is on the road each day. The routing protocol implementation is a first step towards deploying realistic DTN applications on the testbed. In addition, the deployment allows us to study the effect of certain events that are not perfectly modeled in the simulation of our routing protocol. These events include delays caused by computation, wireless channel interference, and operating system delays.

Each bus in DieselNet carries a small-form desktop computer, 40 GB of storage, and a GPS device. The buses operate a 802.11b radio that scans for other buses 10 times a second and an 802.11b access point (AP) that accepts incoming connections. Once a bus is found, a connection is created to the remote AP. (It is likely that the remote bus then creates a connection to the discovered AP, which our software merges into one connection event.) The connection lasts until the radios are out of range. Burgess et al. [6] describes the DieselNet testbed in more detail.

5.1 Deployment

Buses in DieselNet send messages using PROTOCOL RAPID in Section 3, computing the metadata as described in Section 4.2. We generated packets of size 1 KB periodically on each bus with an exponential inter-arrival time. The destinations of the packets included only buses that were scheduled to be on the road, which avoided creation of many packets that could never be delivered. We did not provide the buses information about the

Avg. buses scheduled per day	19
Avg. total bytes transfered per day	261.4 MB
Avg. number of meetings per day	147.5
Percentage delivered per day	88%
Avg. packet delivery delay	91.7 min
Meta-data size/ bandwidth	0.002
Meta-data size/ data size	0.017

Table 3: Deployment of Rapid: Average daily statistics

location or route of other buses on the road. We set the default packet generation rate to 4 packets per hour generated by each bus for every other bus on the road; since the number of buses on the road at any time varies, this is the simplest way to express load. For example, when 20 buses are on the road, the default rate is 1,520 packets per hour.

During the experiments, the buses logged packet generation, packet delivery, delivery delay, meta-data size, and the total size of the transfer opportunity. Buses transfered random data after all routing was complete in order to measure the capacity and duration of each transfer opportunity. The logs were periodically uploaded to a central server using open Internet APs found on the road.

5.2 Performance of deployed RAPID

We measured the routing performance of RAPID on the buses from Feb 6, 2007 until May 14, 2007[1]. The measurements are tabulated in Table 3. We exclude holidays and weekends since almost no buses were on the road, leaving 58 days of experiments. RAPID delivered 88% of packets with an average delivery delay of about 91 minutes. We also note that overhead due to meta-data accounts for less than 0.2% of the total available bandwidth and less than 1.7% of the data transmitted.

5.3 Validating trace-driven simulator

In the next section, we evaluate RAPID using a trace-driven simulator. The simulator takes as input a schedule of node meetings, the bandwidth available at each meeting, and a routing algorithm. We validated our simulator by comparing simulation results against the 58-days of measurements from the deployment. In the simulator, we generate packets under the same assumptions as the deployment, using the same parameters for exponentially distributed inter-arrival times.

Figure 3 shows the average delay characteristics of the real system and the simulator. Delays measured using the simulator were averaged over the 30 runs and the error-bars show a 95% confidence interval. From those results and further analysis, we find with 95% confidence that the simulator results are within 1% of the implementation measurement of average delay. The close correlation between system measurement and simulation increases our confidence in the accuracy of the simulator.

[1]The traces are available at `http://traces.cs.umass.edu`.

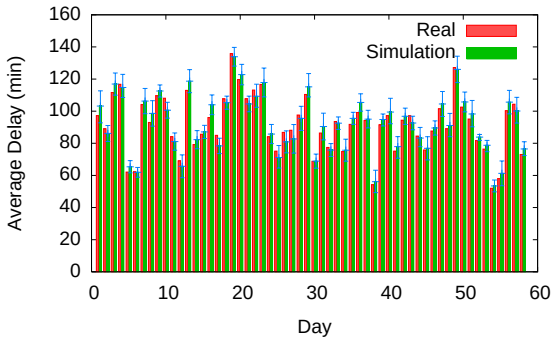

Figure 3: **Trace:** Average delay for 58 days of RAPID real deployment compared to simulation of RAPID using traces

6. EVALUATION

The goal of our evaluation is to show that, unlike existing work, RAPID can improve performance for customizable metrics. We evaluate RAPID using three metrics: minimize maximum delay, minimize average delay, and minimize missed deadlines. In all cases, we found that RAPID significantly outperforms existing protocols and also performs close to optimal for small workloads.

6.1 Experimental setup

Our evaluations are based on a custom event-driven simulator, as described in the previous section. The meeting times between buses in these experiments are not known *a priori*. All values used by RAPID, including average meeting times, are learned during the experiment.

We compare RAPID to five other routing protocols: *MaxProp* [6], *Spray and Wait* [25], *Prophet* [18], *Random*, and *Optimal*. In all experiments, we include the cost of RAPID's in-band control channel for exchanging metadata.

MaxProp operates in a storage- and bandwidth-constrained environment, allows packet replication, and leverages delivery notifications to purge old replicas; of recent related work, it is closest to RAPID's objectives. *Random* replicates randomly chosen packets for the duration of the transfer opportunity. *Spray and Wait* restricts the number of replications of a packets to L, where L is calculated based on the number of nodes in the network. For our simulations, we implemented the binary *Spray and Wait* and set[2] $L = 12$. We implemented *Prophet* with parameters $P_{init} = 0.75$, $\beta = 0.25$ and $\gamma = 0.98$ (parameters based on values used in [18]).

[2]We set this value based on consultation with authors and using LEMMA 4.3 in [25] with $a = 4$.

	Exponential/ Power law	Trace-driven
Number of nodes	20	max of 40
Buffer size	100 KB	40 GB
Transfer opp. size	100 KB	given by trace
Duration	15 min	19 hours each trace
Size of a packet	10 KB	10 KB
Packet generation rate	50 sec mean	1 hour
Delivery deadline	20 sec	2.7 hours

Table 4: Experiment parameters

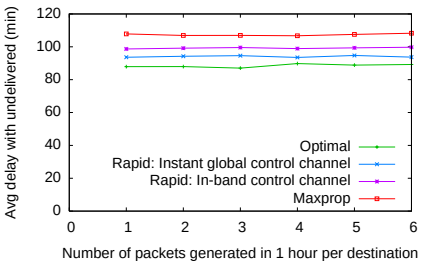

Figure 13: (Trace) Comparison with *Optimal*: Average delay of RAPID is within 10% of *Optimal* for small loads

We also perform experiments where mobility is modeled using a synthetic distribution – in this work we consider exponential and power law distribution. Previous studies [9, 17] have suggested that DTNs among people have a skewed, power law inter-meeting time distribution. The default parameters used for all the experiments are tabulated in Table 4. The parameters for the synthetic mobility model is different from the trace-driven model because the performance between the two models are not comparable.

Each data point is averaged over 10 runs; in the case of trace-driven results, the results are averaged over 58 traces. Each of the 58 days is a separate experiment. In other words, packets that are not delivered by the end of the day are lost. In all experiments, *MaxProp*, RAPID and *Spray and Wait* performed significantly better than *Prophet*, and the latter is not shown in the graphs for clarity.

6.2 Results based on testbed traces

6.2.1 Comparison with existing routing protocols

Our experiments show that RAPID consistently outperforms *MaxProp*, *Spray and Wait* and *Random*. We increased the load in the system up to 40 packets per hour per destination, when *Random* delivers less than 50% of the packets.

Figure 4 shows the average delay of delivered packets using the four protocols for varying loads when RAPID's routing metric is set to minimize average delay (Eq. 1). When using RAPID, the average delay of delivered packets are significantly lower than *MaxProp*, *Spray and Wait* and *Random*. Moreover, RAPID also consistently delivers a greater fraction of packets as shown in Figure 5.

Figure 6 shows RAPID's performance when the routing metric is set to minimize maximum delay (Eq. 3) and similarly Figure 7 shows results when the metric is set to maximize the number of packets delivered within a deadline (Eq. 2).

We note that among *MaxProp*, *Spray and Wait* and *Random*, *MaxProp* delivers the most number of packets, but *Spray and Wait* has marginally lower average delay than *MaxProp*. RAPID significantly outperforms the three protocol for all metrics because of its intentional design.

Standard deviation and similar measures of variance are not appropriate for comparing the mean delays as each bus takes a different geographic route. So, we performed a paired t-test [8] to compare the average delay of every source-destination pair using RAPID to the average delay of the same source-destination pair using *MaxProp* (the second best performing protocol). In our tests, we found p-values always less than 0.0005, indicating the differences between the means reported in these figures are statistically significant.

In a separate experiment (not shown in figure), we find that the number of replications per delivery made by RAPID is 5.2, for a load of 5 packets per hour per destination. For the same load, the number of replications per delivery made by *Random* is 3.5 and *Spray and Wait* is 4.2. We note that we only consider the number of replications for packets that are delivered, and RAPID is set to optimize the average delay metric. Even though it seems that RAPID replicates more aggressively to deliver more packets, RAPID only replicates when bandwidth is available. For example, when the load is increased to 15 packets per hour per destination, the number of replications per delivery made by RAPID reduced to 4.3.

6.2.2 Metadata exchange

We allow RAPID to use as much bandwidth at the start of a transfer opportunity for exchanging metadata as it requires. To see if this approach was wasteful or beneficial, we performed experiments where we limited the total metadata exchanged. Figure 8 shows the average delay performance of RAPID when metadata is limited as a percentage of the total bandwidth. The average delay metric shown here includes the delay for undelivered packets. When a packet is undelivered, it is assumed to be delivered at the end of the day.

The results show that performance increases as the limit is removed and that the best performance results when there is no restriction on metadata at all. The performance of RAPID with complete metadata exchange improves by 20% compared to when no metadata is exchanged. The metadata in this experiment is represented as a percentage of available bandwidth.

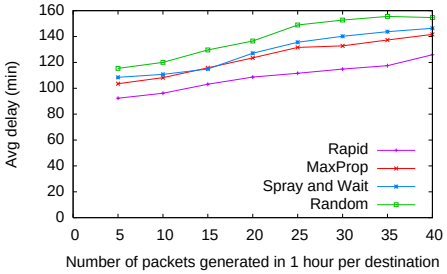

Figure 4: (Trace) Average Delay: RAPID has up to 20% lower delay than *MaxProp* and up to 35% lower delay than *Random*

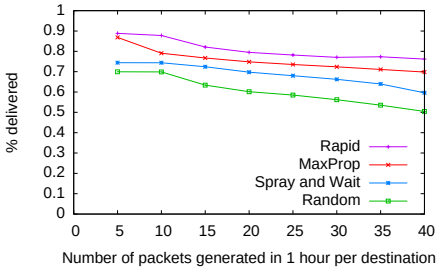

Figure 5: (Trace) Delivery Rate: RAPID delivers up to 14% more than *MaxProp*, 28% than *Spray and Wait* and 45% than *Random*

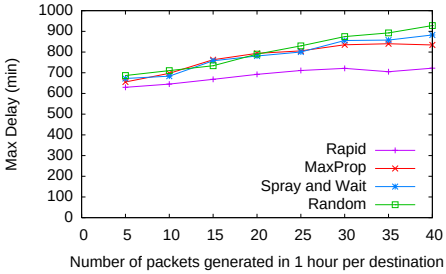

Figure 6: (Trace) Max Delay: Maximum delay of RAPID is up to 90 min lower than *MaxProp*, *Spray and Wait*, and *Random*

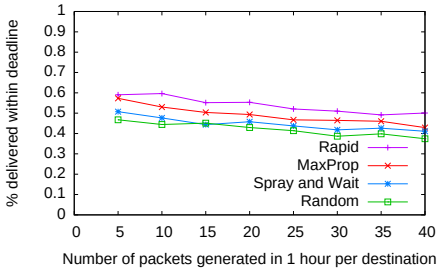

Figure 7: (Trace) Delivery within deadline: RAPID delivers up to 21% more than *MaxProp*, 24% than *Spray and Wait*, 28% than *Random*

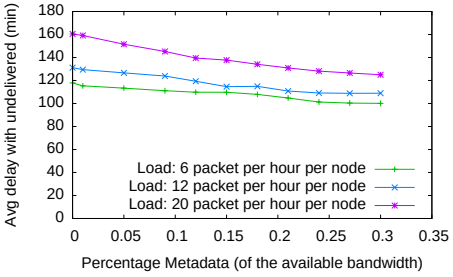

Figure 8: (Trace) Control channel benefit: Average delay performance improves as more metadata is allowed to be exchanged

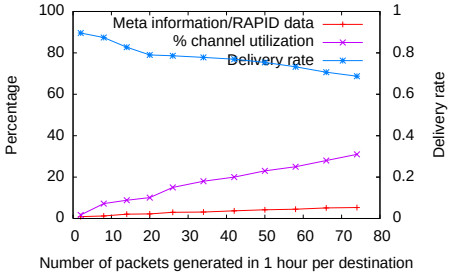

Figure 9: (Trace) Channel utilization: As load increases, delivery rate decreases to 65% but channel utilization is only about 35%

In the next experiment, we analyze total metadata as a percentage of data. In particular, we increase the load to 75 packets per destination per hour to analyze the trend in terms of bandwidth utilization, delivery rate and metadata. Figure 9 shows this trend as load increases. The bandwidth utilization is about 35% for the load of 75 packets per hour per destination, while delivery rate is only about 65%. This suggests that the performance drops even though the network is underutilized, and it is because of the bottleneck links in the network. The available bandwidth varies significantly across transfer opportunities in our bus traces [6].

We also observe that metadata increases to about 4% of data for high loads. This is an order of magnitude higher than the metadata observed as a fraction of bandwidth, again because of the poor channel utilization. The average metadata exchange per contact is proportional to the load and the channel utilization.

RAPID uses more information to improve routing performance. Although the result is intuitive, RAPID uses the additional information to compute packet utilities accurately and in-turn replicate packets intentionally. In contrast, *Spray and Wait* or *Random* cannot use additional information even if available, and *MaxProp* uses additional information only to remove delivered packets [6]. Further, collecting the additional information does not incur a huge overhead in RAPID. The metadata overhead reduces even further with increasing packet size. For example, moving from 1-KB to 10-KB packets reduces RAPID's metadata overhead by an order of magnitude.

There are several scenarios where metadata exchange needs to be limited. For example, when transfer opportunities sizes are much smaller than the number of packets, exchanging all metadata during a transfer opportunity may affect performance. Similarly, since RAPID is a link-state routing protocol, it scales only as well as a link-state protocol. As the network size increases, a node may need to limit the state information it maintains as well as the amount of metadata exchanged. The issue of limiting metadata exchange according to the network scenario will be addressed as part of future work.

6.2.3 *Hybrid DTN with thin continuous connectivity*

In this section, we compare the performance of RAPID using an instant *global* control channel for exchanging metadata as opposed to the default (delayed) *in-band* control channel.

Figure 10 shows the average delay of RAPID when using an in-band control channel compared to a global

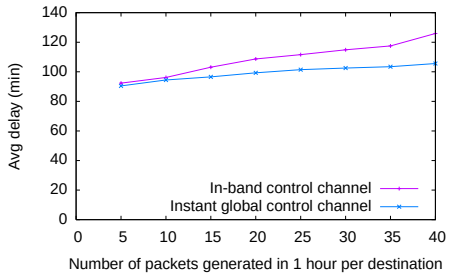

Figure 10: (Trace) Global channel: Average delay of RAPID decreases by up to 20 minutes using instant global control channel

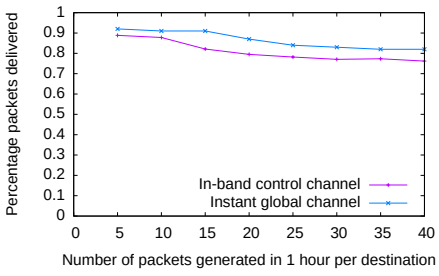

Figure 11: (Trace) Global channel: Delivery rate increases by up to 12% using an instant global control channel, for the average delay metric

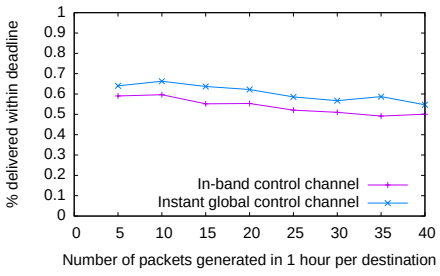

Figure 12: (Trace) Global channel: Packets delivered within deadline increases by about 15% using instant global control channel

channel. We observe that the average delay of delivered packets decreases by up to 20 minutes when using a global channel. For the same experiments, the delivery rate when using an instant global channel increases by up to 12% (shown in Figure 11). Similarly, Figure 12 shows that the percentage packets delivered within a deadline increases by an average of 20% using a global channel. This observation suggests that RAPID's performance can benefit further by using more control information.

One interpretation of the global channel is the use of RAPID as a hybrid DTN where all control traffic goes over a low-bandwidth, long-range radio such as XTend [4]. Since XTend radios support a data rate of about 1 KBps for a range of 1 mile, the radios cannot be used to deliver data packets when the incoming rate is high or packet sizes are large. A hybrid DTN will use a high-cost, low-bandwidth channel for control whenever available and low-cost high-bandwidth delayed channel for data. In our experiments, we assumed that the global channel is instant. While this may not be feasible in practice, the results give an upper bound on RAPID's performance when accurate channel information is available.

6.3 Results compared with Optimal

We compare RAPID to *Optimal*, which is an upper bound on the performance. To obtain the optimal delay, we formulate the DTN routing problem as an Integer Linear Program (ILP) optimization problem when the meeting times between nodes are precisely known (details in a Technical report [3]) and solve the problem using a CPLEX solver [10]. Because the problem grows in complexity with the number of packets, these results are limited to only 6 packets per hour per destination. The ILP objective function minimizes delay of all packets, where the delay of undelivered packets is set to time the packet spent in the system. Accordingly, we add the delay of undelivered packets when presenting the results for RAPID and *MaxProp*.

Figure 13 presents the average delay performance of *Optimal*, RAPID, and *MaxProp*. We observe that for small loads, the performance of RAPID using the in-band control channel is within 10% of the optimum performance, while using *MaxProp* the delays are about 22% from the optimal. RAPID using a global channel performs within 6% of optimal.

6.4 Results from synthetic mobility models

Next, we use an exponential and power law mobility model to compare the performance of RAPID to *MaxProp*, *Random*, and *Spray and Wait*. When mobility is modeled using power law, two nodes meet with an exponential inter-meeting time, but the mean of the exponential distribution is determined by the popularity of the nodes. For the 20 nodes, we randomly set a popularity value of 1 to 20, with 1 being most popular.

We set the mean meeting time for both mobility distribution to 30 seconds. For the power law mobility model, the meeting time is skewed from 30 seconds according to the node's popularity. All other parameters for exponential and power law are identical.

6.4.1 *Powerlaw mobility model, increasing load*

Figure 14 shows the average delay for packets to be delivered (i.e., RAPID is set to use Eq. 1 as a metric). The average delay of packets quickly increase to 20 seconds as load increases in the case of *MaxProp*, *Spray and Wait* and *Random*. In comparison, RAPID's delay does not increase rapidly with increasing load, and is on an average 20% lower than all the three protocols.

Figure 15 shows the performance with respect to minimizing the maximum delay of packets (using Eq. 3 as a metric). RAPID reduces maximum delay by an average of 30% compared to the other protocols. For both the traces and the synthetic mobility, the performance of RAPID is significantly higher than *MaxProp*, *Spray and Wait*, and *Random* for the maximum delay metric. The reason is *MaxProp* prioritizes new packets; older, undelivered packets will not see service as load increases. Similarly, *Spray and Wait* does not give preference to older packets. However, RAPID specifically prioritizes

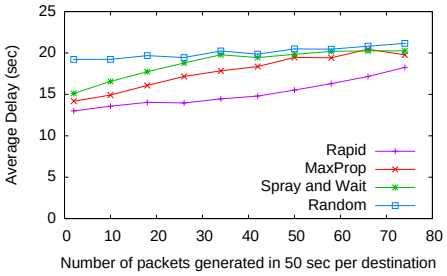

Figure 14: (Powerlaw) Avg Delay: RAPID reduces delay by about 20% compared to *MaxProp*, and 23% than *Spray and Wait* and 25% than *Random*

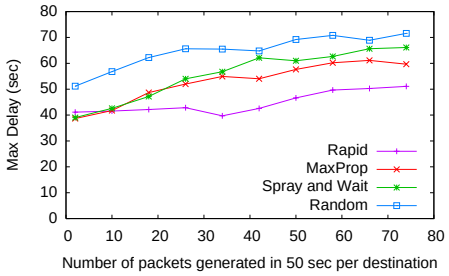

Figure 15: (Powerlaw) Max delay: RAPID's max delay is about 30% lower than *MaxProp*, 35% lower than *Spray and Wait* and 45% lower than *Random*

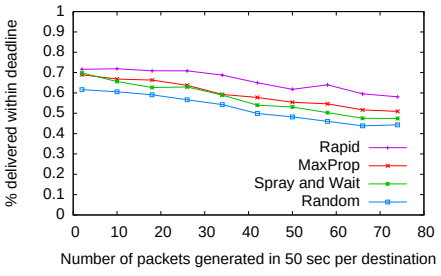

Figure 16: (Powerlaw) Delivery Deadline: RAPID delivers about 20% more packets within deadline when buffer size is constrained, compared to *MaxProp*, and 45% more packets compared to *Spray and Wait* and *Random*

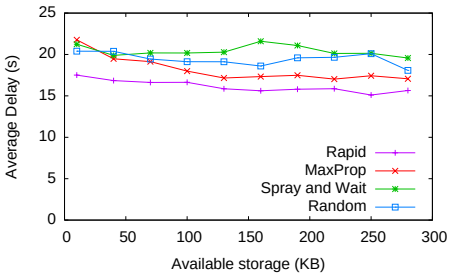

Figure 17: (Powerlaw) Avg Delay with constrained buffer: RAPID reduces average delay by about 23%when buffer size is constrained compared to *MaxProp*, *Spray and Wait* and *Random*

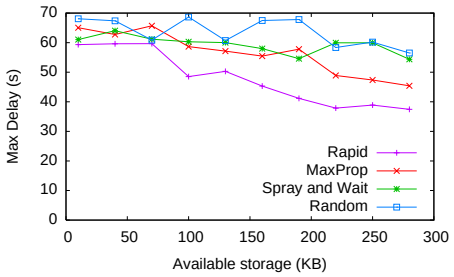

Figure 18: (Powerlaw) Max delay with constrained buffer: RAPID's max delay is about 22% lower than *MaxProp*, 35% lower than *Spray and Wait* and 38% lower than *Random* when buffer is constrained

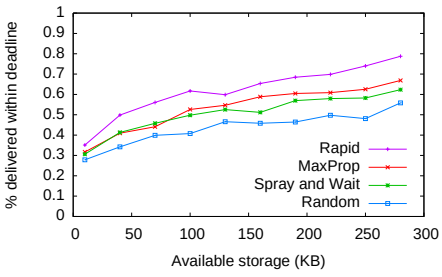

Figure 19: (Powerlaw) Delivery Deadline with constrained buffer: RAPID delivers about 20% more packets within deadline when buffer size is constrained compared to *MaxProp*, and 45% more than *Spray and Wait* and *Random*

older packets to reduce maximum delay.

We observe similar trends in Figure. 16, that shows the performance of the different routing protocols with respect to maximizing the number of packet delivered within an average deadline of 20 sec (RAPID uses Eq. 2).

6.4.2 *Powerlaw mobility model: decreasing storage constraint*

In this set of experiments, we varied available storage from 10 KB to 280 KB and compared the performance of the four routing protocols. We fixed the load to 20 packets per destination and generated packets with a inter-arrival time of 50 seconds.

Figure 17 shows how the average delay of all four protocols vary with increase storage availability. RAPID is able to maintain low delays even when only 10 KB space is available at each node. In comparison, *MaxProp*, *Spray and Wait* and *Random* have an average 23% higher delay.

Figure 18 shows a similar performance trend in terms of minimizing maximum delay. Similar to other experiments, the difference in performance between RAPID and the other three protocols is more marked for the maximum delay metric.

Figure 19 shows how constrained buffers affect the delivery deadline metric. When storage is restricted, *MaxProp* deletes packets that are replicated most number of times, while *Spray and Wait* and *Random* deletes packets randomly. RAPID, when set to maximizing number of packets delivered within a deadline, deletes packets that are most likely to miss the deadline. RAPID is able to best manage limited buffers to deliver packets within a deadline and improves delivery performance by 12% compared to the second-best performing protocol. These experiments suggest that RAPID's utility-driven approach adapts well to storage restrictions as well. We observed similar trends for increasing storage restrictions when using exponential mobility model (not shown in figure).

7. DISCUSSION

1. Not surprising that with more information, rapid performs better.

As a general comment, the paper should express the limitations of RAPID better: the authors perform a quite extensive comparison against a variety of routing protocols for different traffic loads. They do not, however, cover larger messages (10 KB seems unnecessarily small) – which would benefit their data to overhead ratio further (as they note for 10KB to 1KB messages). More importantly, their entire evaluation is limited to a small scale scenario where the suggested metadata exchange and state maintenance is feasible. The authors should at least provide some intuition for larger networks: even if one would limit this to public transport scenarios, most city bus networks will have ten or more times the number of buses in operation. These are clear limitations and they should be clearly stated, e.g., in the discussion/conclusion in the end. Currently, the final picture appears a bit too general.

We agree. We added a discussions section to address this concern. HOW TO DO THIS. ACCEPT THAT FOR SMALLER MESSAGES, THE META DATA IS UNACCEPTABLE.

ADD A DISCUSSIONS SECTION

2) The suggestion with the low rate Xtend background channel warrants careful consideration given the message sizes the authors have discussed: at this size, a control channel (or a cellular network or even MMS) could easily be used to distribute this data. This would also argue for larger message sizes where such alternatives become infeasible. This might be a worth a statement or two.

8. CONCLUSIONS

Previous work in DTN routing protocols has seen only incidental performance improvement from various routing mechanisms and protocol design choices. In contrast, we have proposed a routing protocol for DTNs that intentionally optimizes a specific routing metric by treating DTN routing as a resource allocation problem. Although our approach is heuristic-based, we have proven that an online DTN routing protocol without future knowledge can perform arbitrarily far from optimal. We have also proven that optimally solving the DTN routing problem even with complete knowledge is NP-hard. Our deployment of RAPID in a DTN testbed illustrates that our approach is realistic and effective. We have shown through trace-driven simulations using 65 days of testbed measurements that RAPID yields significant performance gains over previous work.

Acknowledgments

We thank Mark Corner, John Burgess, and Brian Lynn for helping build and maintain DieselNet, Ramgopal Mettu for helping develop the NP-hardness proof, and Erik Learned-Miller and Jérémie Leguay for feedback on earlier drafts. We thank Karthik Thyagarajan for his help in formulating the Integer Linear Program. This research was supported in part by National Science Foundation awards NSF-0133055 and CNS-0519881, CNS-0721779, CNS-0845855.

9. REFERENCES

[1] One laptop per child. `http://www.laptop.org`.
[2] TIER Project, UC Berkeley. `http://tier.cs.berkeley.edu/`.
[3] A. Balasubramanian, B. N. Levine, and A. Venkataramani. DTN Routing as a Resource Allocation Problem. Technical Report 07-37, UMass Amherst, 2007.
[4] N. Banerjee, M. D. Corner, and B. N. Levine. An Energy-Efficient Architecture for DTN Throwboxes. In *Proc. IEEE Infocom*, May 2007.

[5] J. Burgess, G. Bissias, M. D. Corner, and B. N. Levine. Surviving Attacks on Disruption-Tolerant Networks without Authentication. In *Proc. ACM Mobihoc*, September 2007.
[6] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine. MaxProp: Routing for Vehicle-Based Disruption- Tolerant Networks. In *Proc. IEEE Infocom*, April 2006.
[7] B. Burns, O. Brock, and B. N. Levine. *MV* Routing and Capacity Building in Disruption Tolerant Networks. In *Proc. IEEE Infocom*, pages 398–408, March 2005.
[8] G. Casella and R. L. Berger. *Statistical Inference*. Second Edition. Duxbury, 2002.
[9] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott. Impact of Human Mobility on the Design of Opportunistic Forwarding Algorithms. In *Proc. IEEE Infocom*, May 2006.
[10] CPLEX. `http://www.ilog.com`.
[11] J. Davis, A. Fagg, and B. N. Levine. Wearable Computers and Packet Transport Mechanisms in Highly Partitioned Ad hoc Networks. In *Proc. IEEE ISWC*, pages 141–148, October 2001.
[12] N. Garg, S. Sobti, J. Lai, F. Zheng, K. Li, A. Krishnamurthy, and R. Wang. Bridging the Digital Divide. *ACM Trans. on Storage*, 1(2):246–275, May 2005.
[13] B. Hull et al. CarTel: A Distributed Mobile Sensor Computing System. In *Proc. ACM SenSys*, pages 125–138, Oct. 2006.
[14] S. Jain, M. Demmer, R. Patra, and K. Fall. Using Redundancy to Cope with Failures in a Delay Tolerant Network. In *Proc. ACM Sigcomm*, pages 109–120, 2005.
[15] S. Jain, K. Fall, and R. Patra. Routing in a Delay Tolerant Network. In *Proc. ACM Sigcomm*, pages 145–158, Aug. 2004.
[16] E. Jones, L. Li, and P. Ward. Practical Routing in Delay-Tolerant Networks. In *Proc. ACM Chants Workshop*, pages 237–243, Aug. 2005.
[17] J. Leguay, T. Friedman, and V. Conan. DTN Routing in a Mobility Pattern Space. In *Proc. ACM Chants Workshop*, pages 276–283, Aug. 2005.
[18] A. Lindgren, A. Doria, and O. Schelén. Probabilistic Routing in Intermittently Connected Networks. In *Proc. SAPIR Workshop*, pages 239–254, Aug. 2004.
[19] A. Maffei, K. Fall, and D. Chayes. Ocean Instrument Internet. In *Proc. AGU Ocean Sciences Conf.*, Feb 2006.
[20] W. Mitchener and A. Vadhat. Epidemic Routing for Partially Connected Ad hoc Networks. Technical Report CS-2000-06, Duke Univ., 2000.
[21] J. Ott and D. Kutscher. A Disconnection-Tolerant Transport for Drive-thru Internet Environments. In *Proc. IEEE INFOCOM*, pages 1849–1862, Mar. 2005.
[22] J. Partan, J. Kurose, and B. N. Levine. A Survey of Practical Issues in Underwater Networks. In *Proc. ACM WUWNet*, pages 17–24, Sept. 2006.
[23] R. C. Shah, S. Roy, S. Jain, and W. Brunette. Data MULEs: Modeling a Three-tier Architecture for Sparse Sensor Networks. In *Proc. IEEE SNPA*, pages 30–41, May 2003.
[24] T. Small and Z. Haas. Resource and Performance Tradeoffs in Delay-Tolerant Wireless Networks. In *Proc. ACM WDTN*, pages 260–267, Aug. 2005.
[25] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks. In *Proc. ACM WDTN*, pages 252–259, Aug. 2005.
[26] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Performance analysis of mobility-assisted routing. In *ACM MobiHoc*, pages 49–60, May 2006.
[27] T. Spyropoulos and K. Psounis and C. Raghavendra. Single-copy Routing in Intermittently Connected Mobile Networks. In *IEEE SECON*, October 2004.
[28] J. Widmer and J.-Y. Le Boudec. Network Coding for Efficient Communication in Extreme Networks. In *Proc. ACM WDTN*, pages 284–291, Aug. 2005.
[29] Y.-C. Tseng and S.-Y. Ni and Y.-S. Chen and J.-P. Sheu. The Broadcast Storm Problem in a Mobile Ad hoc Network. *Springer Wireless Networks*, 8(2/3):153–167, 2002.
[30] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi. Hardware Design Experiences in ZebraNet. In *Proc. ACM SenSys*, pages 227–238, Nov. 2004.
[31] X. Zhang, G. Neglia, J. Kurose, and D. Towsley. Performance Modeling of Epidemic Routing. In *Proc. IFIP Networking*, May 2006.

APPENDIX

Delay estimation based on dependency graphs:

In Section 3, we presented ESTIMATE_DELAY that estimates expected delays of packets based on the packet's position in a node's buffer. The algorithm ignores some dependencies between packets across node buffers. In this section, we present an algorithm DAG_DELAY to estimate expected delays more accurately without ignoring non vertical dependancies.

To formalize the dependancies, we introduce some notation. Let $G = (V, E)$ be a graph representing a markov network with vertices $V = \{V_1 \cup V_2 \cup \ldots \cup V_m\}$ where $V_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,k}\}$ is the set of k replicas of packet i. All packets in V are destined to the same DTN node — recall that we wish to estimate expected delays of packets based on the current state of the network assuming no further replication, so packets destined to other DTN nodes do not affect the delays of packets in V. An edge (or a path) from one vertex to another indicates a dependency between the delivery time distributions of the corresponding packets. The edges are constructed as follows.

- Each replica is connected to its successor, i.e., the replica immediately ahead of it in the current buffer.
- Each replica is connected to all the replicas of its successor at other DTN node buffers.

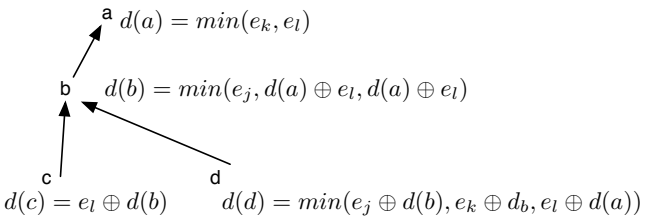

Figure 20: A topologically sorted dependancy graph.

Let the delay distribution of a packet in buffer x be e_x. Let $\oplus$ represent the addition of two distributions (e.g., adding two identical exponential distributions yields a gamma distribution with twice the mean). Assume unit-sized transfer opportunity and packet.

DAG_DELAY first topologically sorts the dependancy graph. For example, the topological sort of Figure 2 is shown in Figure 20. DAG_DELAY computes the delay of the packets in the graph in the topologically

order starting from the top. The information maintained for each of the k replicas $p_1, \ldots, p_k$ of packet p is $\{succ(p_j), e_{vertex(p_j)}\}$, $1 \leq j \leq k$, where $succ(p_j)$ is the successor of the replica p_j, and $vertex(p_j)$ is the DTN buffer where p_j exists.

PROCEDURE DAG_DELAY(p):

1. for each replica $p_j, 1 \leq j \leq k$ of p, do
   (a) Let $s = succ(p_j)$, and $n = vertex(p_j)$
   (b) if $d(s)$ is not defined, then
      i. $d(s) =$DAG_DELAY(s)
   (c) $d'(p_j) = d(s) \oplus e_n$
2. return $d(p) = min(d'(p_1), \ldots, d'(p_k))$

Figure 20 presents the delay of each packet as computed using DAG_DELAY. Although the algorithm is recursive, sequentially computing the delay of packets top down in the DAG and storing the delay values of already computed packets ensures that the delay of each packet is computed exactly once. And since the DAG has no cycles, DAG_DELAY will converge.

DAG_DELAY is an idealized algorithm and its implementation requires a global control channel (introduced in section 6.2.3) with complete knowledge of the system state. Therefore, in our implementation, we use the less accurate but local ESTIMATE_DELAY algorithm. In addition, DAG_DELAY fails when the transfer opportunities are not unit-sized. For example, in Figure 2, if the transfer opportunity and packets are unit-sized, then the delay of packet b depends on the delay of packet a. But if not, then the delay of b may not depend on the delay of a and the dependancy graph is no longer valid. In general, estimating the expected delay of packets is a hard problem even when global knowledge is available but transfer opportunities are *not* unit-sized.

Pathological examples when ESTIMATE_DELAY *fails*

ESTIMATE_DELAY ignores non-vertical edges in the dependency DAG and therefore is not an accurate estimate. Figure 21 shows a pathological case of packet distribution where the estimation error of ESTIMATE_DELAY can be arbitrarily large. In this example, we assume that all packets are destined to node Z. There are $k+1$ replicas of a distributed among nodes $W_1, W_2 \cdots W_k$ and X. There are no replicas of b. We assume that the meeting times between all pairs of nodes is exponentially distributed; further we let the mean meeting time between $W_1, W_2 \cdots W_k$ and Z be λ and between X and Z be $10 \cdot \lambda$. The transfer opportunities are unit-sized.

Algorithm ESTIMATE_DELAY computes the delay of delivery b as the expected time taken for X to meet destination Z twice. Accordingly, a RAPID node estimates the delivery delay of b as an exponential distribution with mean $20 \cdot \lambda$.

DAG_DELAY, on the other hand, takes into account non-vertical dependancies and estimates the delivery

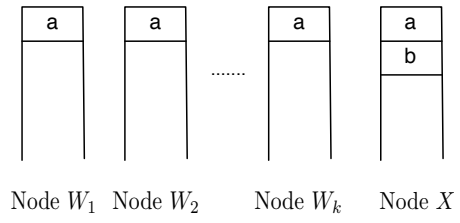

Figure 21: A pathological example of packet distribution among nodes.

delay distribution of b accurately as – the time taken for one of the $k+1$ replicas of a to be delivered followed by X meeting Z. The time taken for a to be delivered is an exponential distribution with mean $\frac{\lambda}{k}$. Accordingly, DAG_DELAY estimates the delivery time of b as a gamma distribution with mean $10 \cdot \lambda + \frac{\lambda}{k}$.

The difference is delay estimation between DAG_DELAY and ESTIMATE_DELAY can be arbitrarily large in this pathological example. But notwithstanding such pathological scenarios, ESTIMATE_DELAY is simple, local, and computationally efficient heuristic to estimate expected delays and we find that it works well in practice.

ILP Formulation

We formulate the DTN routing problem as an ILP to solve it optimally. For the ILP formulation, we assume that the nodes have complete knowledge of future transfer opportunities. To make the formulation tractable, we borrow the idea used in [15] and first divide time into discrete intervals so every node meets at most one other node in an interval. The notations used in the formulation are as follows.

- I: set of time intervals. $t(i)$ is the time interval i ends, and h is the last time interval.
- N: set of nodes
- E: set of edges. An edge is defined when two nodes meeting in an interval. $f(e)$ and $s(e)$, for $e \in E$ are the two vertices incident on the edge. $d(e)$ is the interval in which the edge exists, and $b(e)$ is the bandwidth.When two nodes i and j meet, they are represented by two edges in either direction.
- P: set of packets. *src(p)* and *dest(p)* for $p \in P$ are the source and destination of the packet p. $c(p)$ is the time interval when the packet was created *size(p)* is the size of the packet.

The variables of the ILP are

- $X(p \in P, e \in E) = 1$ if packet p is forwarded over the edge e and is 0 otherwise
- $N(p \in P, n \in N, i \in I) = 1$ if node n has packet p in the interval i and is 0 otherwise
- $D(p \in P, i \in I) = 1$ if packet p is delivered before interval i and is 0 otherwise

X is used to construct the optimal path taken by a

packet. The formulation is:

$$min \sum_{p \in P} \sum_{i \in I} \sum_{e \in E_{(dest(p),i)}} (t(i) - c(p)) \cdot X(p,e) + \sum_{p \in P} (1 - D(p, t(h)) \cdot (t(h) - c(p))$$

All constraints use notations $\forall p, n, i, e$ to mean $\forall p \in P, \forall n \in N, \forall i \in I$ and $\forall e \in E$.

Initialization constraints

$$N(p,n,i) = 0 \ if \ i < c(p) \ \forall p, n, i$$
$$N(p,n,i) = 1 \ if \ src(p) = n \text{ and } c(p) = i \ \forall p$$

Bandwidth constraint

$$\sum_{p \in P} (X(p,e) * size(p) \leq b(e) \ \forall e$$

Transfer constraints

$$N(p,n,i-1) - \sum_{e \in E_{(i,n)}} X(p,e) = 0 \ \forall p, n, i$$
$$\sum_{e \in E_{(n,i)}} X(p,e) - N(p,n,i) = 0 \ \forall p, n, i$$
$$N(p, f(e), d(e) - 1) - X(p,e) >= 0 \ \forall p, e$$

Conservation constraint

$$1 - \sum_{n \in N} N(p,n,i) = 0 \ if \ i > c(p) \ \forall p, e$$

Delivery Constraint

$$D(p,i) - \sum_{e \in E_{(dest(p),.)} : d(e) < i} X(p,e) = 0 \ \forall \ p, i$$

.